

Realizability Toposes from Specifications *

Jonas Frey¹

1 Department of Computer Science
University of Copenhagen, Denmark
jofr@di.ku.dk

Abstract

We investigate a framework of Krivine realizability with I/O effects, and present a method of associating realizability models to *specifications* on the I/O behavior of processes, by using adequate interpretations of the central concepts of *pole* and *proof-like term*. This method does in particular allow to associate realizability models to computable functions.

Following recent work of Streicher and others we show how these models give rise to *triposes* and *toposes*.

1 Introduction

Krivine realizability with side effects has been introduced by Miquel in [14]. In this article we demonstrate how an instance of Miquel’s framework including I/O instructions allows to associate *realizability toposes* to *specifications*, i.e. sets of requirements imposed on the I/O behavior of programs. Since the requirement to compute a specific function f can be viewed as a specification, we do in particular obtain a way to *associate toposes to computable functions*.

These toposes are different from traditional ‘Kleene’ realizability toposes such as the *effective topos* [6] in that we associate toposes to *individual* computable functions, whereas the effective topos incorporates *all* recursive functions on equal footing. Another difference to the toposes based on Kleene realizability is that the internal logic of the latter is *constructive*, whereas the present approach is based on Krivine’s realizability interpretation [10], which validates classical logic.

To represent specifications we make use of the fact that Krivine’s realizability interpretation is parametric over a set of processes called the *pole*. The central observation (Lemma 27 and Theorem 28) is that non-trivial specifications on program behavior give rise to poles leading to consistent (i.e. non-degenerate) interpretations.

To give a categorical account of Krivine realizability we follow recent work of Streicher [18] and others [17, 20, 2], which demonstrates how Krivine realizability models give rise to *triposes*. Toposes are then obtained via the tripos-to-topos construction [7].

Our basic formalism is an extension of the Krivine machine (2) that gives an operational semantics to I/O instructions for single bits. We give two formulations of the operational semantics – one (3) in terms of a transition relation on processes including a *state* (which is adequate for reasoning about function computation), and one (4) in terms of a labeled transition system admitting to reason about program equivalence in terms of bisimulation. The two operational semantics are related by Corollary 7, which we use to prove a Turing completeness result in Theorem 12.

* This work is supported by the Danish Council for Independent Research Sapere Aude grant “Complexity through Logic and Algebra” (COLA).



1.1 Related work

The idea of adding instructions with new evaluation rules to the machine plays a central role in Krivine’s writings, as a means to realize non-logical axioms. Citing from [11]:

“Indeed, when we realize usual axioms of mathematics, we need to introduce, one after the other, the very standard tools in system programming: for the law of Peirce, these are continuations (particularly useful for exceptions); for the axiom of dependent choice, these are the clock and the process numbering; for the ultrafilter axiom and the well ordering of \mathbb{R} , these are no less than I/O instructions on a global memory, in other words assignment.”

Although features like exceptions and memory are often called *effects*, it is arguable whether they should be called *side effects*, since they do not interact with the outside world.

The idea to add instructions for *side effects* which are influenced by – and influence – the outside world, has already been investigated by Miquel [14, Section 2.2], and our execution relation (3) can be viewed as an instance of his framework.

What sets the present approach apart is that Miquel views the state of the world (represented by a forcing condition) as being *part* of a process and requires poles to be saturated w.r.t. all (including effectful) reductions, whereas for us poles are sets of ‘bare’ processes without state, which are saturated only w.r.t. reduction free of side-effects.

This difference is crucial in that it enables the construction of poles from specifications.

2 Syntax and machine

In this section we recall Krivine’s abstract machine with continuations as described in [10]. We then go on to describe an extension of the syntax by I/O instructions, and describe an operational semantics as a transition relation on triples (p, ι, o) of process, input, and output.

2.1 Krivine’s machine

We recall the underlying syntax and machine of Krivine’s classical realizability from [10]. The syntax consists of three syntactic classes called *terms*, *stacks*, and *processes*.

$$\begin{array}{lll}
 \text{Terms:} & t ::= x \mid \lambda x.t \mid tt \mid \alpha \mid k_\pi & \\
 \text{Stacks:} & \pi ::= \pi_0 \mid t \cdot \pi & t \text{ closed, } \pi_0 \in \Pi_0 \\
 \text{Processes:} & p ::= t \star \pi & t \text{ closed}
 \end{array} \tag{1}$$

Thus, the terms are the terms of the λ -calculus, augmented by a constant α for call/cc, and continuation terms k_π for any stack π . A stack, in turn, is a list of closed terms terminated by an element π_0 of a designated set Π_0 of *stack constants*. A process is a pair $t \star \pi$ of a closed term and a stack. The set of closed terms is denoted by Λ , the set of stacks is Π , and the set of processes is $\Lambda \star \Pi$.

Krivine’s machine is now defined by a transition relation \succ on processes called *evaluation*.

$$\begin{array}{lll}
 (\text{push}) & tu \star \pi & \succ \quad t \star u \cdot \pi \\
 (\text{pop}) & (\lambda x.t[x]) \star u \cdot \pi & \succ \quad t[u] \star \pi \\
 (\text{save}) & \alpha \star t \cdot \pi & \succ \quad t \star k_\pi \cdot \pi \\
 (\text{restore}) & k_\pi \star t \cdot \rho & \succ \quad t \star \pi
 \end{array} \tag{2}$$

The first two rules implement *weak head reduction* of λ -terms, and the third and fourth rule capture and restore continuations.

2.2 The machine with I/O

To incorporate I/O we modify the syntax as follows:

Terms:	$t ::= x \mid \lambda x.t \mid tt \mid \mathbf{c} \mid \mathbf{k}_\pi \mid r \mid \mathbf{w}0 \mid \mathbf{w}1 \mid \mathbf{end}$	
Stacks:	$\pi ::= \varepsilon \mid t \cdot \pi$	t closed
Processes:	$p ::= t \star \pi \mid \top$	t closed

The grammar for terms is extended by constants $r, \mathbf{w}0, \mathbf{w}1, \mathbf{end}$ for reading, writing and termination, and in exchange the stack constants are omitted – ε is the empty stack. Finally there is a *process constant* \top also representing termination – the presence of both \mathbf{end} and \top will be important in Section 3.

We write Λ_e and Π_e for the sets of terms and stacks of the syntax with I/O, and P for the set of processes. Furthermore, we denote by Λ_p the set of *pure* terms, i.e. terms not containing any of $r, \mathbf{w}0, \mathbf{w}1, \mathbf{end}$.

The operational semantics of the extended syntax is given in terms of *execution contexts*, which are triples (p, ι, o) of a process p , and a pair $\iota, o \in \{0, 1\}^*$ of binary strings representing input and output. On these execution contexts, we define the *execution relation* \rightsquigarrow as follows:

$$\begin{array}{llll}
 (\tau) & (t \star \pi, \iota, o) & \rightsquigarrow & (u \star \rho, \iota, o) \quad \text{whenever } t \star \pi \succ u \star \rho \\
 (r0) & (r \star t \cdot u \cdot v \cdot \pi, 0\iota, o) & \rightsquigarrow & (t \star \pi, \iota, o) \\
 (r1) & (r \star t \cdot u \cdot v \cdot \pi, 1\iota, o) & \rightsquigarrow & (u \star \pi, \iota, o) \\
 (r\varepsilon) & (r \star t \cdot u \cdot v \cdot \pi, \varepsilon, o) & \rightsquigarrow & (v \star \pi, \varepsilon, o) \\
 (\mathbf{w}0) & (\mathbf{w}0 \star t \cdot \pi, \iota, o) & \rightsquigarrow & (t \star \pi, \iota, 0o) \\
 (\mathbf{w}1) & (\mathbf{w}1 \star t \cdot \pi, \iota, o) & \rightsquigarrow & (t \star \pi, \iota, 1o) \\
 (\mathbf{e}) & (\mathbf{end} \star \pi, \iota, o) & \rightsquigarrow & (\top, \iota, o)
 \end{array} \tag{3}$$

Thus, if there is neither of $r, \mathbf{w}0, \mathbf{w}1, \mathbf{end}$ in head position, the process is reduced as in (2) without changing ι and o . If r is in head position, the computation selects one of the first three arguments depending on whether the input starts with a 0, a 1, or is empty. $\mathbf{w}0$ and $\mathbf{w}1$ write out 0 and 1, and \mathbf{end} discards the stack and yields \top , which represents successful termination.

We observe that the execution relation is *deterministic*, i.e. for every execution context there is at most one transition possible, which is determined by the term in head position, and in case of r also by the input.

2.3 Representing functions

We view the above formalism as a model of computation that explicitly includes reading of input, and writing of output.

Consequently, when thinking about expressivity we are not so much interested in the ability of the machine to transform abstract representations of data like ‘Church numerals’, but rather in the functions on binary strings that processes can compute by reading their argument from the input, and writing the result to the output.

► **Definition 1.** For $n \in \mathbb{N}$, $\text{bin}(n) \in \{0, 1\}^*$ is the base 2 representation of n . 0 is represented by the empty string, thus we have e.g. $\text{bin}(0) = \varepsilon$, $\text{bin}(1) = 1$, $\text{bin}(2) = 10$, $\text{bin}(3) = 11$, ...

A process p is said to *implement* a partial function $f : \mathbb{N} \rightarrow \mathbb{N}$, if $(p, \text{bin}(n), \varepsilon) \rightsquigarrow^* (\top, \varepsilon, \text{bin}(f(n)))$ for all $n \in \text{dom}(f)$.

► **Remark.** There is a stronger version of the previous definition which requires $(p, \text{bin}(n), \varepsilon)$ to diverge or block for $n \notin \text{dom}(f)$, and a completeness result like Thm. 12 can be shown for the strengthened definition as well.

We use the weaker version, since we expect the poles \perp_f defined in Section 5.2.1 to be better behaved this way.

2.4 β -reduction

To talk about contraction of single β -redexes which are not necessarily in head position in a process p , we define *contexts* – which are terms/stacks/processes with a single designated hole $[\cdot]$ in term position – by the following grammar:

$$\begin{array}{lll} \text{Term contexts:} & t[\cdot] & ::= [\cdot] \mid \lambda x.t[\cdot] \mid t[\cdot]t \mid tt[\cdot] \mid \mathbf{k}_{\pi[\cdot]} \\ \text{Stack contexts:} & \pi[\cdot] & ::= t \cdot \pi[\cdot] \mid t[\cdot] \cdot \pi \quad t, t[\cdot] \text{ closed} \\ \text{Process contexts:} & p[\cdot] & ::= t[\cdot] \star \pi \mid t \star \pi[\cdot] \end{array}$$

Contexts are used to talk about substitution that allows capturing of variables – as described in [1, 2.1.18], given a context $t[\cdot]/\pi[\cdot]/p[\cdot]$ and a term u , $t[u]/\pi[u]/p[u]$ is the result of replacing the hole $[\cdot]$ in $t[\cdot]/\pi[\cdot]/p[\cdot]$ by u , allowing potential free variables in u to be captured. We say that u is *admissible* for $t[\cdot]/\pi[\cdot]/p[\cdot]$, if $t[u]/\pi[u]/p[u]$ is a valid term/stack/process conforming to the closedness condition for terms making up stacks.

Now we can express β -reduction as the action of contracting a single redex: given a redex $(\lambda x.u)v$ which is admissible for a context $t[\cdot]/\pi[\cdot]/p[\cdot]$, we have

$$t[(\lambda x.u)v] \rightarrow_{\beta} t[u[v/x]] \quad \pi[(\lambda x.u)v] \rightarrow_{\beta} \pi[u[v/x]] \quad p[(\lambda x.u)v] \rightarrow_{\beta} p[u[v/x]],$$

and any single β -reduction can uniquely be written this way. β -equivalence \simeq_{β} is the equivalence relation generated by β -reduction.

3 Bisimulation and \top -equivalence

To reason efficiently about execution of processes with side effects – in particular to show Turing completeness in Section 4 – we want to show that although the computation model imposes a deterministic reduction strategy, we can perform β -reduction anywhere in a process without changing its I/O behavior.

The natural choice of concept to capture ‘equivalence of I/O behavior’ is *weak bisimilarity* (see [13, Section 4.2]), and in order to make this applicable to processes we have to reformulate the operational semantics as a *labeled transition system* (LTS).

We use the set $\mathcal{L} = \{r0, r1, r\varepsilon, w0, w1, e\}$ of labels, where $r0, r1$ represent reading of a 0 or 1, respectively, and $w0, w1$ represent writing of bits. $r\varepsilon$ represents the unsuccessful attempt of reading on empty input, and e represents successful termination. The set $Act = \mathcal{L} \cup \{\tau\}$ of *actions* contains the labels as well as the symbol τ representing a ‘silent’ transition, that is used to represent effect-free evaluation.

The transition system on processes is now given as follows.

$$\begin{array}{lll} (\lambda x.t[x]) \star t \cdot \pi & \xrightarrow{\tau} & t[u] \star \pi \quad r \star t \cdot u \cdot v \cdot \pi \xrightarrow{r0} t \star \pi \quad w0 \star t \cdot \pi \xrightarrow{w0} t \star \pi \\ tu \star \pi & \xrightarrow{\tau} & t \star u \cdot \pi \quad r \star t \cdot u \cdot v \cdot \pi \xrightarrow{r1} u \star \pi \quad w1 \star t \cdot \pi \xrightarrow{w1} t \star \pi \\ \mathbf{c} \star t \cdot \pi & \xrightarrow{\tau} & t \star \mathbf{k}_{\pi} \cdot \pi \quad r \star t \cdot u \cdot v \cdot \pi \xrightarrow{r\varepsilon} v \star \pi \quad \mathbf{end} \star \pi \xrightarrow{e} \top \\ \mathbf{k}_{\pi} \star t \cdot \rho & \xrightarrow{\tau} & t \star \pi \end{array} \quad (4)$$

Observe that the τ -transitions are in correspondence with the transitions of the evaluation relation (2), and the labeled transitions correspond to the remaining transitions of the execution relation (3).

We now recall the definition of *weak bisimulation relation* from [13, Section 4.2].

► **Definition 2.**

- For processes p, q we write $p \xrightarrow{\tau} q$ for $p \xrightarrow{\tau}^* q$, and for $\alpha \neq \tau$ we write $p \xrightarrow{\alpha} q$ for $\exists p', q'. p \xrightarrow{\tau} p' \xrightarrow{\alpha} q' \xrightarrow{\tau} q$.
- A *weak bisimulation* on P is a binary relation $R \subseteq P^2$ such that for all $\alpha \in Act$ and $(p, q) \in R$ we have

$$\begin{aligned} p \xrightarrow{\alpha} p' &\Rightarrow \exists q'. q \xrightarrow{\alpha} q' \wedge (p', q') \in R \quad \text{and} \\ q \xrightarrow{\alpha} q' &\Rightarrow \exists p'. p \xrightarrow{\alpha} p' \wedge (p', q') \in R. \end{aligned} \tag{5}$$

- Two processes p, q are called *weakly bisimilar* (written $p \approx q$), if there exists a weak bisimulation relation R with $(p, q) \in R$.

We recall the following important properties of the weak bisimilarity relation \approx .

► **Lemma 3.** *Weak bisimilarity is itself a weak bisimulation, and furthermore it is an equivalence relation.*

Proof. [13, Proposition 4.2.7] ◀

To show that β -equivalent processes are bisimilar, we have to find a bisimulation relation containing β -equivalence. The following relation does the job.

► **Definition 4** (γ -equivalence). γ -equivalence (written $p \simeq_{\gamma} q$) is the equivalence relation on processes that is generated by β -reduction and τ -transitions.

► **Lemma 5.** *γ -equivalence of processes is a weak bisimulation.*

Proof. It is sufficient to verify conditions (5) on the generators of γ -equivalence, i.e. one-step β -reductions and τ -transitions. Therefore we show the following:

1. if $p \xrightarrow{\tau} q$ and $p \xrightarrow{\alpha} p'$ then there exists q' with $q \xrightarrow{\alpha} q'$ and $p' \simeq_{\gamma} q'$
2. if $p \xrightarrow{\tau} q$ and $q \xrightarrow{\alpha} q'$ then there exists p' with $p \xrightarrow{\alpha} p'$ and $p' \simeq_{\gamma} q'$
3. if $p \rightarrow_{\beta} q$ and $p \xrightarrow{\alpha} p'$ then there exists q' with $q \xrightarrow{\alpha} q'$ and $p' \simeq_{\gamma} q'$
4. if $p \rightarrow_{\beta} q$ and $q \xrightarrow{\alpha} q'$ then there exists p' with $p \xrightarrow{\alpha} p'$ and $p' \simeq_{\gamma} q'$

In the first case, the fact that the LTS can only branch if r is in head position, and this does not involve τ -transitions, implies that $\alpha = \tau$ and $p' = q$, and we can choose $q' = q$ as well. In the second case we have $p \xrightarrow{\alpha} q'$ and thus can choose $p' = q'$.

For cases 3 and 4, which we treat simultaneously, we have to analyze the structure of p and q , which are of the form $r[(\lambda x. s)t]$ and $r[s[t/x]]$ for some context $r[\cdot]$ (see Section 2.4). The proof proceeds by cases on the structure of $r[\cdot]$.

If $r[\cdot]$ is of either of the forms $(st \star \pi)[\cdot]^1$, $\mathbf{w0} \star \pi[\cdot]$, $\mathbf{wl} \star \pi[\cdot]$, $\mathbf{c} \star \pi[\cdot]$, $(k_{\pi} \star \rho)[\cdot]$, or $\mathbf{end} \star \pi[\cdot]$, then it is immediately evident that p and q can perform the same unique transition (if any), and the results will again be β -equivalent (possibly trivially, since the redex can get deleted in the transition).

If $r[\cdot]$ is of the form $((\lambda y. u) \star \pi)[\cdot]$ then this is true as well, regardless of whether the hole is in u or in π (here the redex can be duplicated, if the hole is in the first term in π).

If $r[\cdot]$ is of the form $r \star \pi[\cdot]$ then several transitions may be possible, but any transition taken by either of p or q can also be taken by the other, and the results will again be β -equivalent.

¹ The notation is meant to convey that we don't care if the hole is in s , t , or π .

It remains to consider $r[\cdot]$ of the form $[\cdot] \star \pi$. In this case, $p = (\lambda x . s)t \star \pi$ and $q = s[t/x] \star \pi$. Here p can perform the transition $p \xrightarrow{\tau} (\lambda x . s) \star t \star \pi$ which can be matched by $q \xrightarrow{\tau} q$ where we have $(\lambda x . s) \star t \star \pi \simeq_{\gamma} p \simeq_{\gamma} q$. In the other direction we have $p \xrightarrow{\alpha} q'$ for every $q \xrightarrow{\alpha} q'$ since $p \xrightarrow{\tau} q$. \blacktriangleleft

The following definition and corollary makes the link between the execution relation (3) and the LTS (4).

► **Definition 6.** Two execution contexts (p, ι, o) , (q, ι', o') are called \top -equivalent (written $(p, \iota, o) \sim_{\top} (q, \iota', o')$), if for all $\iota'', o'' \in \{0, 1\}^*$ we have

$$(p, \iota, o) \rightsquigarrow^* (\top, \iota'', o'') \quad \text{iff} \quad (q, \iota', o') \rightsquigarrow^* (\top, \iota'', o'').$$

► **Corollary 7.**

1. $p \approx q$ implies $(p, \iota, o) \sim_{\top} (q, \iota, o)$ for all $\iota, o \in \{0, 1\}^*$.
2. $(p, \iota, o) \rightsquigarrow^* (q, \iota', o')$ implies $(p, \iota, o) \sim_{\top} (q, \iota', o')$.
3. $(p, \iota, o) \sim_{\top} (\top, \iota', o')$ implies $(p, \iota, o) \rightsquigarrow^* (\top, \iota', o')$.

Proof. For the first claim we show that

$$p \approx q, \quad (p, \iota, o) \rightsquigarrow^* (\top, \iota', o') \quad \text{implies} \quad (q, \iota, o) \rightsquigarrow^* (\top, \iota', o')$$

by induction on the length of $(p, \iota, o) \rightsquigarrow^* (\top, \iota', o')$. The base case is clear. For the induction step assume that $(p, \iota, o) \rightsquigarrow (p^*, \iota^*, o^*) \rightsquigarrow^* (\top, \iota', o')$. If the initial transition is a (τ) in the execution relation (3), then have $p^* \cong q$, $\iota^* = \iota$ and $o^* = o$, and we can apply the induction hypothesis. If the initial transition corresponds to another clause in (3), then there is a corresponding transition $p \xrightarrow{\alpha} q$ with $\alpha \in \mathcal{L}$ in the LTS (4), and by bisimilarity there exists a q^* with $q \xrightarrow{\alpha} q^*$ and $p^* \approx q^*$. Now the induction hypothesis implies $(q^*, \iota^*, o^*) \rightsquigarrow^* (\top, \iota', o')$, and from $q \xrightarrow{\alpha} q^*$ we can deduce $(q, \iota, o) \rightsquigarrow^* (q^*, \iota^*, o^*)$ by cases on α .

The second claim follows since \rightsquigarrow is deterministic.

The third claim follows since (\top, ι', o') can not perform any more transitions. \blacktriangleleft

4 Expressivity

In this section we show that the machine with I/O is Turing complete, i.e. that every computable $f : \mathbb{N} \rightarrow \mathbb{N}$ can be implemented in the sense of Def. 1 by a process p .

Roughly speaking, given f , we define a process p that reads the input, transforms it into a Church numeral, applies a term t that computes f on the level of Church numerals, and then writes the result out.

To decompose the task we define terms R and W for reading and writing, with the properties that $(R * \pi, \text{bin}(n), o) \sim_{\top} (\bar{n} * \pi, \varepsilon, o)$ (\bar{n} is the n -th Church numeral), and $(W \bar{n} * \pi, \iota, \varepsilon) \sim_{\top} (\top, \iota, \text{bin}(n))$ for all $n \in \mathbb{N}$.

Now the naive first attempt to combine R and W with the term t computing the function would be something like $W(tR)$, but this would only work if the operational semantics was call by value. The solution is to use Krivine's *storage operators* [9] which were devised precisely to simulate call by value in call by name, and we use a variation of them.

The following definition introduces the terms R and W , after giving some auxiliary definitions.

► **Definition 8.** E, Z, B, C, H, Y, S are λ -terms satisfying

$$\begin{array}{lll} B \bar{n} \simeq_\beta \overline{2n} & E \overline{(2n)} s t & \simeq_\beta s \quad Y t \simeq_\beta t (Y t) \\ C \bar{n} \simeq_\beta \overline{2n+1} & E \overline{(2n+1)} s t & \simeq_\beta t \\ H \bar{n} \simeq_\beta \overline{\text{floor}(n/2)} & Z \overline{(0)} s t & \simeq_\beta s \\ S \bar{n} \simeq_\beta \overline{n+1} & Z \overline{(n+1)} s t & \simeq_\beta t \end{array}$$

for all terms s, t and $n \in \mathbb{N}$, where \bar{n} is the Church numeral $\lambda f x. f^n x$.²

The terms F, R, W are defined as follows:

$$\begin{aligned} F &= \lambda h y. h(S y)^3 \\ R &= Y Q \bar{0} \quad \text{where} \quad Q = \lambda x n. r(x(B n))(x(C n))n \\ W &= Y V \quad \text{where} \quad V = \lambda x n. Z n \text{end}(E n(\mathbf{w}0 x(H n))(\mathbf{w}1 x(H n))) \end{aligned}$$

The next three lemmas explain the roles of the terms R, F , and W .

► **Lemma 9.** For all $n \in \mathbb{N}$, $\pi \in \Pi$ and $o \in \{0, 1\}^*$ we have $(R \star \pi, \text{bin}(n), o) \sim_\top (\bar{n} \star \pi, \varepsilon, o)$.

Proof. For all $n \in \mathbb{N}$ we have $Y Q \bar{n} \simeq_\beta Q(Y Q) \bar{n} \simeq_\beta r(Y Q \overline{(2n)})(Y Q \overline{(2n+1)}) \bar{n}$, and thus

$$\begin{aligned} (Y Q \bar{n} \star \pi, \varepsilon, o) &\sim_\top (\bar{n} \star \pi, \varepsilon, o) \\ (Y Q \bar{n} \star \pi, 0\iota, o) &\sim_\top (Y Q \overline{(2n)} \star \pi, \iota, o) \\ (Y Q \bar{n} \star \pi, 1\iota, o) &\sim_\top (Y Q \overline{(2n+1)} \star \pi, \iota, o) \end{aligned}$$

The claim follows by induction on the length of $\text{bin}(n)$, since $\text{bin}(2n) = \text{bin}(n)0$ for $n > 0$, and $\text{bin}(2n+1) = \text{bin}(n)1$. ◀

► **Lemma 10.** For $n \in \mathbb{N}$ and t any closed term, we have $\bar{n} F t \bar{0} \simeq_\beta t \bar{n}$.

Proof. This is because $\bar{n} F t \bar{0} \simeq_\beta F^n t \bar{0} \simeq_\beta t (S^n \bar{0}) \simeq_\beta t \bar{n}$, where the second step can be shown by induction on n . ◀

► **Lemma 11.** For all $n \in \mathbb{N}$, $\pi \in \Pi$ and $\iota \in \{0, 1\}^*$ we have $(W \bar{n} \star \pi, \iota, \varepsilon) \sim_\top (\top, \iota, \text{bin}(n))$.

Proof. We have $W \bar{n} \simeq_\beta V W \bar{n} \simeq_\beta Z \bar{n} \text{end}(E \bar{n}(\mathbf{w}0 W(H \bar{n}))(\mathbf{w}1 W(H \bar{n})))$, and therefore

$$\begin{aligned} (W \bar{0} \star \pi, \iota, o) &\sim_\top (\text{end} \star \pi, \iota, o) \sim_\top (\top, \iota, o) \\ (W \overline{(2n)} \star \pi, \iota, o) &\sim_\top (\mathbf{w}0 W(H \overline{(2n)}) \star \pi, \iota, o) \sim_\top (W(\bar{n}) \star \pi, \iota, 0o) \quad \text{for } (n > 0) \\ (W \overline{(2n+1)} \star \pi, \iota, o) &\sim_\top (\mathbf{w}1 W(H \overline{(2n+1)}) \star \pi, \iota, o) \sim_\top (W(\bar{n}) \star \pi, \iota, 1o). \end{aligned}$$

The claim follows again by induction on the length of $\text{bin}(n)$. ◀

► **Theorem 12.** Every computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ can be implemented by a process p .

Proof. From [5, Thm. 4.23] we know that there exists a term t with $t \bar{n} \simeq_\beta \overline{f(n)}$ for $n \in \text{dom}(f)$. The process p is given by $R \star F \cdot t \cdot \bar{0} \cdot F \cdot W \cdot \bar{0}$. Indeed, for $n \in \text{dom}(f)$ we have

$$\begin{aligned} (R \star F \cdot t \cdot \bar{0} \cdot F \cdot W \cdot \bar{0}, \text{bin}(n), \varepsilon) &\sim_\top (\bar{n} \star F \cdot t \cdot \bar{0} \cdot F \cdot W \cdot \bar{0}, \varepsilon, \varepsilon) \sim_\top (\bar{n} F t \bar{0} \star F \cdot W \cdot \bar{0}, \varepsilon, \varepsilon) \\ &\sim_\top (t \bar{n} \star F \cdot W \cdot \bar{0}, \varepsilon, \varepsilon) \sim_\top (\overline{f(n)} \star F \cdot W \cdot \bar{0}, \varepsilon, \varepsilon) \\ &\sim_\top (W \overline{f(n)} \star \varepsilon, \varepsilon, \varepsilon) \sim_\top (\top, \varepsilon, \text{bin}(f(n))) \end{aligned}$$

and we deduce $(R \star F \cdot t \cdot \bar{0} \cdot F \cdot W \cdot \bar{0}, \text{bin}(n), \varepsilon) \rightsquigarrow^* (\top, \varepsilon, \text{bin}(f(n)))$ by Corollary 7-3. ◀

² Such terms exist by elementary λ -calculus, see e.g. [5, Chapters 3,4]. In particular, Y is known as *fixed point operator*.

³ This is (part of) a *storage operator* for Church numerals [9].

5 Realizability and triposes

The aim of this section is to describe how the presence of I/O instructions allows to define new realizability models, which we do in the categorical language of triposes and toposes [21].

In Subsection 5.1 we give a categorical reading of Krivine’s realizability interpretation as described in [10] and show how it gives rise to triposes. In Subsection 5.2 we show how the definitions can be adapted to the syntax and machine with I/O, and how this allows us to define new realizability models from specifications.

The interpretation of Krivine realizability in terms of triposes is due to Streicher [18], and has further been explored in [2]. However, the presentation here is more straightforward since the constructions and proofs do not rely on *ordered combinatory algebras*, but directly rephrase Krivine’s constructions categorically.

5.1 Krivine’s classical realizability

Throughout this subsection we work with the syntax (1) without I/O instructions but with stack constants.

Krivine’s realizability interpretation is always given relative to a set of processes called a ‘pole’ – the choice of pole determines the interpretation.

► **Definition 13.** A *pole* is a set $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$ of processes which is *saturated*, in the sense that $p \in \perp\!\!\!\perp$ and $p' \succ p$ implies $p' \in \perp\!\!\!\perp$.

As Miquel [15] demonstrated, the pole can be seen as playing the role of the parameter R in Friedman’s negative translation [3]. In the following we assume that a pole $\perp\!\!\!\perp$ is fixed.

A *truth value* is by definition a set $S \subseteq \Pi$ of stacks. Given a truth value S and a term t , we write $t \Vdash S$ – and say ‘ t realizes S ’ – if $\forall \pi \in S. t \star \pi \in \perp\!\!\!\perp$. We write $S^\perp = \{t \in \Lambda \mid t \Vdash S\}$ for the set of realizers of Π . So unlike in Kleene realizability the elements of a truth value are not its realizers – they should rather be seen as ‘refutations’, and indeed larger subsets of Π represent ‘falsier’ truth values⁴; in particular falsity is defined as

$$\perp = \Pi.$$

Given truth values $S, T \subseteq \Pi$, we define the implication $S \Rightarrow T$ as follows.

$$S \Rightarrow T = S^\perp \cdot T = \{s \cdot \pi \mid s \Vdash S, \pi \in T\}$$

With these definitions we can formulate the following lemma, which relates refutations of a truth value S with realizers of its negation.

► **Lemma 14.** *Given $\pi \in S \subseteq \Pi$, we have $k_\pi \Vdash S \Rightarrow \perp$.*

Proof. We have to show that $k_\pi \star t \cdot \rho \in \perp\!\!\!\perp$ for all $t \Vdash S$ and $\rho \in \Pi$. This is because $k_\pi \star t \cdot \rho \succ t \star \pi$, where $\pi \in S$ and $t \Vdash S$. ◀

A (*semantic*) *predicate* on a set I is a function $\varphi : I \rightarrow P(\Pi)$ from I to truth values. On semantic predicates we define the basic logical operations of falsity, implication, universal

⁴ For this reason, Miquel [15, 16] calls the elements of $P(\Pi)$ *falsity values*.

quantification, and reindexing by

$$\begin{aligned}
\perp(i) &= \Pi && (\text{falsity}) \\
(\varphi \Rightarrow \psi)(i) &= \varphi(i) \Rightarrow \psi(i) = \varphi(i)^{\perp} \cdot \psi(i) && (\text{implication}) \\
\forall_f(\theta)(i) &= \bigcup_{f(j)=i} \theta(j) && (\text{universal quantification}) \\
f^*\varphi &= \varphi \circ f && (\text{reindexing})
\end{aligned} \tag{6}$$

for $\varphi, \psi : I \rightarrow P(\Pi)$, $\theta : J \rightarrow P(\Pi)$ and $f : J \rightarrow I$. Thus, for any function $f : J \rightarrow I$, the function \forall_f (called ‘universal quantification along f ’) maps predicates on J to predicates on I ⁵, and the function f^* (called ‘reindexing along f ’) maps predicates on I to predicate on J . We write \forall_I for universal quantification along the terminal projection $I \rightarrow 1$.

Next, we come to the concept of ‘truth/validity’ of the interpretation. We can not simply call a truth value ‘true’ if it has a realizer – this would lead to inconsistency as soon as the pole \perp is nonempty, since $k_\pi t \Vdash \perp$ for any process $t \star \pi \in \perp$. The solution is to single out a set PL of ‘well-behaved’ realizers called ‘proof-like terms’. We recall the definition from [10].

► **Definition 15.** The set $\text{PL} \subseteq \Lambda$ of *proof-like terms* is the set of terms t that do not contain any continuations k_π .

As Krivine [10, pg. 2] points out, t is a proof-like term if and only if it does not contain any stack constant $\pi_0 \in \Pi_0$ (since continuation terms k_π necessarily contain a stack constant at the end of π , and conversely stacks can only occur as continuations in a term).

Proof-like terms give us a concept of logical validity – a truth value S is called *valid*, if there exists a proof-like term t with $t \Vdash S$.

With this notion, we are ready to define the centerpiece of the realizability model, which is the *entailment relation* on predicates.

► **Definition 16.** For any set I and integer n , the $(n+1)$ -ary entailment relation (\vdash_I) on predicates on I is defined by

$$\varphi_1 \dots \varphi_n \vdash_I \psi \quad \text{if and only if} \quad \exists t \in \text{PL} . t \Vdash \forall_I(\varphi_1 \Rightarrow \dots \Rightarrow \varphi_n \Rightarrow \psi).$$

If the right hand side proposition holds, we call t a *realizer* of $\varphi_1 \dots \varphi_n \vdash_I \psi$.

Thus, $\varphi_1 \dots \varphi_n \vdash_I \psi$ means that the truth value $\forall_I(\varphi_1 \Rightarrow \dots \Rightarrow \varphi_n \Rightarrow \psi)$ is valid. More explicitly this can be written out as

$$\exists t \in \text{PL} \forall i \in I, u_1 \in \varphi_1(i)^{\perp}, \dots, u_n \in \varphi_n(i)^{\perp}, \pi \in \psi(i) . t \star u_1 \dots u_n \cdot \pi \in \perp.$$

With the aim to show that the semantic predicates form a tripos in Theorem 22, we now prove that the entailment ordering models the logical rules in Table (1). The first eight rules form a standard natural deduction system for (the \perp, \Rightarrow fragment of) classical propositional logic, but for universal quantification we give categorically inspired rules that bring us quicker to where we want, and in particular avoid having to deal with variables.

► **Lemma 17.** *The rules displayed in Table 1 are admissible for the entailment relation, in the sense that if the hypotheses hold then so does the conclusion.*

⁵ The usual $\forall x : A$ from predicate logic corresponds to taking f to be a projection map $\pi_1 : \Gamma \times A \rightarrow \Gamma$, see e.g. [8, Chapter 4].

$\frac{}{\varphi \vdash_I \varphi} \text{ (Ax)}$	$\frac{\Gamma \vdash_I \perp}{\Gamma \vdash_I \psi} \text{ (}\perp\text{E)}$
$\frac{\Gamma, \varphi \vdash_I \psi}{\Gamma \vdash_I \varphi \Rightarrow \psi} \text{ (}\Rightarrow\text{I)}$	$\frac{\Gamma \vdash_I \psi \quad \Delta \vdash_I \psi \Rightarrow \theta}{\Gamma, \Delta \vdash_I \theta} \text{ (}\Rightarrow\text{E)}$
$\frac{\Gamma \vdash_I \psi}{\sigma(\Gamma) \vdash_I \psi} \text{ (S)}$	$\frac{}{\Delta \mid \Gamma \vdash_I ((\psi \Rightarrow \perp) \Rightarrow \psi) \Rightarrow \psi} \text{ (PeL)}$
$\frac{\Gamma \vdash_I \psi}{A, \Gamma \vdash_I \psi} \text{ (W)}$	$\frac{A, A, \Gamma \vdash_I \psi}{A, \Gamma \vdash_I \psi} \text{ (C)}$
$\frac{f^* \Gamma \vdash_J \xi}{\Gamma \vdash_I \forall_f \xi} \text{ (}\forall\text{I)}$	$\frac{\Gamma \vdash_I \forall_f \xi}{f^* \Gamma \vdash_J \xi} \text{ (}\forall\text{E)}$

φ, ψ, θ are predicates on I , i.e. functions $I \rightarrow P(\Pi)$, and $\Gamma \equiv \varphi_1 \dots \varphi_n$ and $\Delta \equiv \psi_1 \dots \psi_m$ are lists of such predicates. ξ is a predicate on J , and $f : J \rightarrow I$ is a function. σ is a permutation of $\{1, \dots, n\}$. $f^* \Gamma$ is an abbreviation for $f^* \varphi_1 \dots f^* \varphi_n$, and $\sigma(\Gamma)$ is an abbreviation for $\varphi_{\sigma(1)} \dots \varphi_{\sigma(n)}$.

■ **Table 1** Admissible rules for the entailment relation.

Proof. (Ax) rule: The conclusion is realized by $\lambda x. x$.

(\perp E) rule: every realizer of the hypothesis is also a realizer of the conclusion, since $\psi(i) \subseteq \perp(i) = \Pi$ for all $i \in I$.

(\Rightarrow I) rule: the hypothesis and the conclusion have precisely the same realizers.

(\Rightarrow E) rule: if t realizes $\Delta \vdash_I \psi \Rightarrow \theta$ and u realizes $\Gamma \vdash_i \psi$ then $\Gamma, \Delta \vdash_i \theta$ is realized by $\lambda x_1 \dots x_n y_1 \dots y_m. t y_1 \dots y_m (u x_1 \dots x_n)$.

(PeL) rule ('Peirce's law'): the conclusion is realized by α . To see this, let $i \in I$, $t \Vdash (\psi(i) \Rightarrow \perp) \Rightarrow \psi(i)$, and $\pi \in \psi(i)$. Then we have $\alpha \star t \cdot \pi \succ t \star k_\pi \cdot \pi$, which is in \perp since $k_\pi \cdot \pi \in (\psi(i) \Rightarrow \perp) \Rightarrow \psi(i)$ by Lemma 14 and the definition (6) of implication.

(W) rule: if t realizes $\Gamma \vdash_I \psi$, then $\lambda x. t$ realizes $A, \Gamma \vdash_I \psi$.

(C) rule: if t realizes $A, A, \Gamma \vdash_I \psi$, then $\lambda x. t x x$ realizes $A, \Gamma \vdash_I \psi$.

(S) rule: if t realizes $\Gamma \vdash_I \psi$, then $\lambda x_{\sigma(1)} \dots x_{\sigma(n)}. t x_1 \dots x_n$ realizes $\sigma(\Gamma) \vdash_I \psi$.

(\forall I) and (\forall E) rules: $\Gamma \vdash_I \forall_f \xi$ and $f^* \Gamma \vdash_J \xi$ have exactly the same realizers. Indeed, a realizer of $f^* \Gamma \vdash_J \xi$ is a term t satisfying

$$\forall j \in J, u_1 \in \varphi_1(f(j))^\perp, \dots, u_n \in \varphi_n(f(j))^\perp, \pi \in \xi(j). t \star u_1 \dots u_n \cdot \pi \in \perp,$$

and a realizer of $\Gamma \vdash_I \forall_f \xi$ is a term t satisfying

$$\forall i \in I, u_1 \in \varphi_1(i)^\perp, \dots, u_n \in \varphi_n(i)^\perp, \pi \in \bigcup_{f(j)=i} \xi(j). t \star u_1 \dots u_n \cdot \pi \in \perp,$$

and both statements can be rephrased as a quantification over pairs (i, j) with $f(j) = i$. ◀

We only defined the propositional connectives \perp, \Rightarrow , since \top, \wedge, \vee, \neg can be encoded as follows:

$$\begin{aligned} \top &\equiv \perp \Rightarrow \perp & \neg \varphi &\equiv \varphi \Rightarrow \perp \\ \varphi \wedge \psi &\equiv (\varphi \Rightarrow (\psi \Rightarrow \perp)) \Rightarrow \perp & \varphi \vee \psi &\equiv (\varphi \Rightarrow \perp) \Rightarrow \psi \end{aligned} \quad (7)$$

With these encodings it is routine to show the following.

► **Lemma 18.** *With the connectives \top, \wedge, \vee, \neg encoded as in (7), the rules of propositional classical natural deduction (e.g. system Nc in [19, Section 2.1.8]) are derivable from the rules in Table 1.*

With this we can show that for any set I , the binary part of the entailment relation makes $P(\Pi)^I$ into a *Boolean prealgebra*.

► **Definition 19.** A *Boolean prealgebra* is a preorder (B, \leq) which

1. has binary *joins* and *meets* – denoted by $x \vee y$ and $x \wedge y$ for $x, y \in D$,
2. has a *least element* \perp and a *greatest element* \top ,
3. is *distributive* in the sense that $x \wedge (y \vee z) \cong (x \wedge y) \vee (x \wedge z)$ for all $x, y, z \in B$, and
4. is *complemented*, i.e. for every $x \in D$ there exists a $\neg x$ with $x \wedge \neg x \cong \perp$ and $x \vee \neg x \cong \top$.

► **Lemma 20.** *Writing $\varphi \leq \psi$ for $\varphi \vdash_I \psi$, $(P(\Pi)^I, \leq)$ is a Boolean prealgebra.*

Proof. The (Ax) rule implies that \leq is reflexive, and transitivity follows from the derivation

$$\frac{\varphi \vdash_I \psi \quad \frac{\psi \vdash_I \theta}{\vdash_I \psi \Rightarrow \theta}}{\varphi \vdash_I \theta}$$

Thus, \leq is a preorder on $P(\Pi)^I$.

The joins, meets, complements, and least and greatest element are given by the corresponding logical operations as defined in (6) and (7).

The required properties all follow from derivability of corresponding entailments and rules in classical natural deduction – for example, $\varphi \wedge \psi$ is a binary meet of φ and ψ since

$$(*) \text{ the entailments } \varphi \wedge \psi \vdash_I \varphi \text{ and } \varphi \wedge \psi \vdash_I \psi \text{ and the rule } \frac{\theta \vdash_I \varphi \quad \theta \vdash_I \psi}{\theta \vdash_I \varphi \wedge \psi}$$

are derivable.

Distributivity follows from derivability of the entailments $\varphi \wedge (\psi \vee \theta) \vdash_I (\varphi \wedge \psi) \vee (\varphi \wedge \theta)$ and $(\varphi \wedge \psi) \vee (\varphi \wedge \theta) \vdash_I \varphi \wedge (\psi \vee \theta)$. ◀

We now come to *triposes*, which are a kind categorical model for higher order logic. We use a ‘strictified’ version of the original definition [7, Def. 1.2] since this bypasses some subtleties and is sufficient for our purposes. Furthermore, we are only interested modeling classical logic here, and thus can restrict attention to triposes whose fibers are Boolean (instead of Heyting) prealgebras.

► **Definition 21.** A *strict⁶ Boolean tripos* is a contravariant functor $\mathcal{P} : \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Ord}$ from the category of sets to the category of preorders such that

- for every set I , the preorder $\mathcal{P}(I)$ is a Boolean prealgebra, and for any function $f : J \rightarrow I$, the induced monotone map $\mathcal{P}(f) : \mathcal{P}(I) \rightarrow \mathcal{P}(J)$ preserves all Boolean prealgebra structure.
- for any $f : J \rightarrow I$, $\mathcal{P}(f)$ has left and right adjoints⁷ $\exists_f \dashv \mathcal{P}(f) \dashv \forall_f$ such that

$$\text{for any pullback square}^8 \quad \begin{array}{ccc} L & \xrightarrow{q} & K \\ p \downarrow & & \downarrow g \\ J & \xrightarrow{f} & I \end{array} \quad (8)$$

⁶ ‘Strict’ refers to the facts that (i) \mathcal{P} is a *functor*, not merely a pseudofunctor (ii) the Boolean prealgebra structure is preserved ‘on the nose’ by the monotone maps $\mathcal{P}(f)$ (iii) the Beck-Chevalley condition is required up to equality, not merely isomorphism, (iv) we require equality and uniqueness in the last condition. Every strict tripos is a tripos in the usual sense, and conversely it can be shown that any tripos is equivalent to a strict one.

⁷ ‘Adjoint’ in the sense of ‘adjoint functor’, where monotone maps are viewed as functors between degenerate categories.

- we have $\mathcal{P}(g) \circ \forall_f = \forall_q \circ \mathcal{P}(p)$ (this is the *Beck-Chevalley condition*), and
- there exists a *generic predicate*, i.e. a set \mathbf{Prop} and an element $\mathbf{tr} \in \mathcal{P}(\mathbf{Prop})$ such that for every set I and $\varphi \in \mathcal{P}(I)$ there exists a unique function $f : I \rightarrow \mathbf{Prop}$ with $\mathcal{P}(f)(\mathbf{tr}) = \varphi$.

The assignment $I \mapsto (P(\Pi)^I, \leq)$ extends to a functor $\mathcal{P}_{\perp} : \mathbf{Set}^{\mathbf{op}} \rightarrow \mathbf{Ord}$ by letting $\mathcal{P}_{\perp}(f) = f^*$, i.e. mapping every function $f : J \rightarrow I$ to the reindexing function along f , which is monotone since every realizer of $\varphi \vdash_I \psi$ is also a realizer of $\varphi \circ f \vdash_J \psi \circ f$.

► **Theorem 22.** \mathcal{P}_{\perp} is a strict Boolean tripos.

Proof. We have shown in Lemma 20 that the preorders $(P(\Pi)^I, \leq)$ are Boolean prealgebras. It is immediate from (6) that the reindexing functions f^* preserve \perp and \Rightarrow , and the other Boolean operations are preserved since they are given by encodings.

The identity function $\text{id} : P(\Pi) \rightarrow P(\Pi)$ is a generic predicate for \mathcal{P}_{\perp} .

The $(\forall I)$ and $(\forall E)$ rules together imply that the operation $\forall_f : P(\Pi)^I \rightarrow P(\Pi)^J$ is right adjoint to f^* for any $f : J \rightarrow I$. Existential quantification along $f : J \rightarrow I$ is given by $\exists_f = \neg \circ \forall_f \circ \neg$, which is left adjoint to f^* since

$$\neg \forall_f \neg \varphi \vdash_I \psi \quad \text{iff} \quad \neg \psi \vdash_I \forall_f \neg \varphi \quad \text{iff} \quad f^* \neg \psi \vdash_J \neg \varphi \quad \text{iff} \quad \neg f^* \psi \vdash_J \neg \varphi \quad \text{iff} \quad \varphi \vdash_J f^* \psi$$

for all $\varphi : J \rightarrow P(\Pi)$ and $\psi : I \rightarrow P(\Pi)$.

It remains to verify the Beck-Chevalley condition. Given a square as in (8) we have

$$g^* \forall_f (\varphi(k)) = \bigcup \{ \varphi(j) \mid f(j) = g(k) \} \quad \text{and} \quad \forall_q (p^*(k)) = \bigcup \{ \varphi(j) \mid \exists l. pl = j \wedge ql = k \},$$

and the two terms are equal since the square is a pullback. ◀

Thus we obtain a tripos \mathcal{P}_{\perp} for each pole \perp . As Hyland, Johnstone, and Pitts showed in [7], every tripos \mathcal{P} gives rise to a topos $\mathbf{Set}[\mathcal{P}]$ via the *tripos-to-topos construction*. Since the fibers of the triposes \mathcal{P}_{\perp} are Boolean prealgebras, the toposes $\mathbf{Set}[\mathcal{P}_{\perp}]$ are Boolean as well, which means that their internal logic is classical.

5.1.1 Consistency

Triposes of the form \mathcal{P}_{\perp} can be degenerate in two ways: if \perp is empty then $\mathcal{P}_{\perp}(I) \simeq (P(I), \subseteq)$ for every set I , and the topos $\mathbf{Set}[\mathcal{P}_{\perp}]$ is equivalent to the category \mathbf{Set} .

If, in the other extreme, the pole is so big that there exists a proof-like t realizing \perp , i.e. falsity is valid in the model, then we have $\mathcal{P}_{\perp}(I) \simeq 1$ for all I (since t realizes every entailment $\varphi \vdash_I \psi$), and the topos $\mathbf{Set}[\mathcal{P}_{\perp}]$ is equivalent to the terminal category.

By *consistency* we mean that falsity is *not* valid, or equivalently that

$$\forall t \in \mathbf{PL} \exists \pi \in \Pi. t \star \pi \notin \perp. \tag{9}$$

The ‘canonical’ (according to Krivine [12]) non-trivial consistent pole is the *thread model*, which is given by postulating a stack constant π_t for each proof-like term t , and defining $\perp = \{ p \in \Lambda \star \Pi \mid \neg \exists t \in \mathbf{PL}. t \star \pi_t \rightsquigarrow^* p \}$. Then the processes $t \star \pi_t$ are not in \perp for any proof-like t , which ensures the validity of condition (9).

In the next section we show how the presence of side effects allows to define a variety of new, ‘meaningful’ consistent poles.

⁸ The square being a pullback means that $f \circ p = g \circ q$ and $\forall jk. f(j) = g(k) \Rightarrow \exists l. pl = j \wedge ql = k$.

5.2 Krivine realizability with I/O

The developments of the previous section generalize pretty much directly to the syntax with I/O. Concretely, we carry over the definitions of *pole*, *truth value*, *realizer*, *predicate*, and of the basic logical operations $\perp, \Rightarrow, \forall$, by replacing Λ with Λ_e , Π with Π_e , and $\Lambda \star \Pi$ with P .

We point out that in presence of effects, Definition 13 only means that \perp is saturated w.r.t. *effect-free* evaluation, in contrast to Miquel's approach [14] where a pole is a set of (what we call) execution contexts, closed under the entire execution relation.

The concept of proof-like term deserves some reexamination. It turns out that the appropriate concept of *proof-like term* is 'term not containing any side effects'. This is consistent with Definition 15 if we read 'free of side effects' as 'free of non-logical constructs', which are the stack constants in Krivine's case. Continuation terms, on the other hand, can be considered proof-like. We redefine therefore:

► **Definition 23.** The set $PL \subseteq \Lambda_e$ of *proof-like terms* is the set of terms not containing any of the constants $r, w0, w1, end$.

With this rephrased definition of proof-like term, we can define the entailment relation on the extended predicates in the same way:

► **Definition 24.** For any set I and integer n , the $(n + 1)$ -ary entailment relation (\vdash_I) on the set $P(\Pi_e)^I$ of extended predicates on I is defined by

$$\varphi_1 \dots \varphi_n \vdash_I \psi \quad \text{if and only if} \quad \exists t \in PL. t \Vdash \forall_I (\varphi_1 \Rightarrow \dots \Rightarrow \varphi_n \Rightarrow \psi).$$

As a special case, the ordering on extended predicates is defined by

$$\varphi \leq \psi \quad \text{if and only if} \quad \exists t \in PL. t \Vdash \forall_I (\varphi \Rightarrow \psi).$$

With these definitions, we can state analogues of Lemma 20 and Theorem 22:

► **Theorem 25.**

- For each set I , the order $(P(\Pi_e)^I, \leq)$ of extended predicates is a Boolean prealgebra.
- The assignment $I \mapsto (P(\Pi_e)^I, \leq)$ gives rise to a strict Boolean tripos $\mathcal{P}_\perp : \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Ord}$.

Proof. This follows from the arguments in Section 5.1, since the proofs of Lemmas 14, 17, 18, 20, and of Theorem 22 are not obstructed in any way by the new constants, nor do they rely on stack constants. The redefinition of 'proof-like term' does not cause any problems either, since we never relied on proof-like terms not containing continuation terms. ◀

The above rephrasing of the definition of proof-like term admits an intuitive reformulation of the consistency criterion (9):

► **Lemma 26.** A pole \perp is consistent if and only if every $p \in \perp \setminus \{\top\}$ contains a non-logical constant, i.e. one of $r, w0, w1, end$.

Proof. If every element of $p \in \perp \setminus \{\top\}$ contains a non-logical constant, then $t \star \varepsilon$ is not in \perp for any proof-like t , which implies (9).

On the other hand, if $t \star \pi \in \perp$ does not contain any non-logical constant then $k_\pi t$ is a proof-like term which realizes \perp , since for any $\rho \in \Pi_e$ we have $k_\pi t \star \rho \succ k_\pi \star t \cdot \rho \succ t \star \pi \in \perp$. ◀

5.2.1 Poles from specifications

The connection between poles and specifications is established by the following lemma.

► **Lemma 27.** *Every set $\perp\!\!\!\perp$ of processes that is closed under weak bisimilarity is a pole.*

Proof. This is because $p \approx q$ whenever $p \succ q$, which follows from Lemma 5. ◀

Since we can assume that for any reasonable specification the processes implementing it are closed under weak bisimilarity, we can thus conclude that for any specification, the set of processes implementing it is a pole. For example:

- $\perp\!\!\!\perp_{cp}$ is the set of processes that read the input, copy every bit immediately to the output, and terminate when the input is empty. We have $Y \star (\lambda x. r(\mathbf{w0} x)(\mathbf{w1} x)\mathbf{end}) \in \perp\!\!\!\perp_{cp}$.
- $\perp\!\!\!\perp_{cp'}$ contains the processes that first read the entire input, and then write out the same string and terminate. We have $R \star F \cdot W \cdot \bar{0} \in \perp\!\!\!\perp_{cp'}$ with the notations of Section 4.
- For any partial function $f : \mathbb{N} \rightarrow \mathbb{N}$, the pole $\perp\!\!\!\perp_f$ consists of those processes that implement f in the sense of Definition 1.
- Since poles are closed under unions, we can define the pole $\perp\!\!\!\perp_F = \bigcup_{f \in F} \perp\!\!\!\perp_f$ for any set $F \subseteq (\mathbb{N} \rightarrow \mathbb{N})$ of partial functions.

5.2.2 Toposes from computable functions

We are particularly interested in the poles $\perp\!\!\!\perp_f$ associated to computable functions f , and we want to use the associated triposes $\mathcal{P}_f = \mathcal{P}_{\perp\!\!\!\perp_f}$ and toposes $\mathbf{Set}[\mathcal{P}_f]$ to study these functions.

The following theorem provides a first ‘sanity check’, in showing that the associated models are non-degenerate.

► **Theorem 28.** *Let $f : \mathbb{N} \rightarrow \mathbb{N}$.*

- $\perp\!\!\!\perp_f$ *is consistent if and only if f is not totally undefined.*
- $\perp\!\!\!\perp_f$ *is non-empty if and only if f is computable.*

Proof. The first claim follows from Lemma 26. If $n \in \text{dom}(f)$ and $t \star \pi$ implements f , then $(t \star \pi, \text{bin}(n), \varepsilon)$ must terminate and thus $t \star \pi$ must contain an **end** instruction. The totally undefined function, on the other hand, is by definition implemented by every process.

For the second claim, we have shown in Theorem 12 that every computable f is implemented by some process. Conversely, every implementable function is computable since the Krivine machine with I/O is an effective model of computation. ◀

5.3 Discussion and future work

The structure and properties of the toposes $\mathbf{Set}[\mathcal{P}_f]$ remain mysterious for the moment, and in future work we want to explore which kind of properties of f are reflected in $\mathbf{Set}[\mathcal{P}_f]$. In the spirit of Grothendieck [4] we want to view the toposes $\mathbf{Set}[\mathcal{P}_f]$ as *geometric* rather than logical objects, the guiding intuition being that $\mathbf{Set}[\mathcal{P}_f]$ can be seen as representation of ‘the space of solutions to the algorithmic problem of computing f ’, encoding e.g. information on how algorithms computing f can be decomposed into simpler parts.

Evident problems to investigate are to understand the lattice of truth values in $\mathbf{Set}[\mathcal{P}_f]$, and to determine for which pairs f, g of functions the associated toposes are equivalent, and which functions can be separated.

A more audacious goal is to explore whether $\mathbf{Set}[\mathcal{P}_f]$ can teach us something about the complexity of a computable function f . The Krivine machine with I/O seems to be a model of computation that is fine grained enough to recognize and differentiate time complexity

of different implementations of f , but it remains to be seen in how far this information is reflected in the ‘geometry’ of $\mathbf{Set}[\mathcal{P}_f]$.

Acknowledgements

Thanks to Jakob Grue Simonsen and Thomas Streicher for many discussions.

References

- 1 H.P. Barendregt. *The lambda calculus, Its syntax and semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, revised edition, 1984.
- 2 W. Ferrer, J. Frey, M. Guillermo, O. Malherbe, and A. Miquel. Ordered combinatory algebras and realizability. *arXiv preprint arXiv:1410.5034*, 2014.
- 3 H. Friedman. Classically and intuitionistically provably recursive functions. In *Higher set theory (Proc. Conf., Math. Forschungsinst., Oberwolfach, 1977)*, volume 669 of *Lecture Notes in Math.*, pages 21–27. Springer, Berlin, 1978.
- 4 A. Grothendieck, M. Artin, and J.L. Verdier. Théorie des topos et cohomologie étale des schémas. *Lecture Notes in Mathematics*, 269, 1972.
- 5 J. Roger Hindley and Jonathan P. Seldin. *Lambda-calculus and combinators, an introduction*. Cambridge University Press, Cambridge, 2008.
- 6 J.M.E. Hyland. The effective topos. In *The L.E.J. Brouwer Centenary Symposium (Noordwijkerhout, 1981)*, volume 110 of *Stud. Logic Foundations Math.*, pages 165–216. North-Holland, Amsterdam, 1982.
- 7 J.M.E. Hyland, P.T. Johnstone, and A.M. Pitts. Tripos theory. *Math. Proc. Cambridge Philos. Soc.*, 88(2):205–231, 1980.
- 8 B. Jacobs. *Categorical logic and type theory*. Elsevier Science Ltd, 2001.
- 9 J.L. Krivine. Lambda-calcul, évaluation paresseuse et mise en mémoire. *RAIRO Inform. Théor. Appl.*, 25(1):67–84, 1991.
- 10 J.L. Krivine. Realizability in classical logic. *Panoramas et synthèses*, 27:197–229, 2009.
- 11 J.L. Krivine. Realizability algebras: a program to well order \mathbb{R} . *Log. Methods Comput. Sci.*, 7(3):3:02, 47, 2011.
- 12 J.L. Krivine. Realizability algebras II: New models of ZF + DC. *Log. Methods Comput. Sci.*, 8(1):1:10, 28, 2012.
- 13 R. Milner. Operational and algebraic semantics of concurrent processes. In *Handbook of theoretical computer science, Vol. B*, pages 1201–1242. Elsevier, Amsterdam, 1990.
- 14 A. Miquel. Classical modal realizability and side effects. *preprint*, 2009.
- 15 A. Miquel. Existential witness extraction in classical realizability and via a negative translation. *Log. Methods Comput. Sci.*, 7(2):2:2, 47, 2011.
- 16 A. Miquel. Forcing as a program transformation. In *26th Annual IEEE Symposium on Logic in Computer Science—LICS 2011*, pages 197–206. IEEE Computer Soc., Los Alamitos, CA, 2011.
- 17 W.P. Stekelenburg. *Realizability Categories*. PhD thesis, Utrecht University, 2013.
- 18 T. Streicher. Krivine’s classical realisability from a categorical perspective. *Mathematical Structures in Computer Science*, 23(06):1234–1256, 2013.
- 19 A.S. Troelstra and H. Schwichtenberg. *Basic proof theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1996.
- 20 J. van Oosten. *Classical Realizability*. Invited talk at “Cambridge Category Theory Seminar”, slides at <http://www.staff.science.uu.nl/~ooste110/talks/cambr060312.pdf>.
- 21 J. van Oosten. *Realizability: An Introduction to its Categorical Side*. Elsevier Science Ltd, 2008.